Matt Horton

CS470 Homework 4

February 20, 2019

Problem 5.9 on page 251

Race conditions are possible in many computer systems. Consider a banking system that maintains an account balance with two functions: deposit(amount) and withdraw(amount). These two functions are passed the amount that is to be deposited or withdrawn from the bank account balance. Assume that a husband and wife share a bank account. Concurrently, the husband calls the withdraw() function and the wife calls deposit(). Describe how a race condition is possible and what might be done to prevent the race condition from occurring.

If there is not enough for a withdrawal and it hangs, then the deposit function is unable to access the account to make more funds available.

Allow the account to go into a negative balance.

Declining a withdrawal request sets a flag that states the next operation must be a deposit

Problem 5.10 on page 251. Note that Figure 5.25 is on page 252.

The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, P0 and P1, share the following variables:

boolean flag[2]; /\* initially false \*/

int turn;

The structure of process Pi (i == 0 or 1) is shown in Figure 5.25. The other process is Pj (j == 1 or 0). Prove that the algorithm satisfies all three requirements for the critical-section problem.

The Pj cannot acquire its own mutex as Pi has set it, and it has also verified that Pi is using its mutex. Adding n mutex’s adds the necessary level of control for n processes.

Problem 5.22 on page 254.

Consider the code example for allocating and releasing processes shown in Figure 5.27.

1. Identify the race condition(s).

If something bumps number\_of\_processes over the MAX before the conditional

Or maybe number of processes could go negative if release is called a disproportionate amount of times

1. Assume you have a mutex lock named mutex with the operations acquire() and release(). Indicate where the locking needs to be placed to prevent the race condition(s).

Right before the ++numer\_of\_processes and right before the –number\_of\_processes

c. Could we replace the integer variable

int number of processes = 0

with the atomic integer

atomic t number of processes = 0

to prevent the race condition(s)?

yes

Problem 5.23 on page 254.

Servers can be designed to limit the number of open connections. For example, a server may wish to have only N socket connections at any point in time. As soon as N connections are made, the server will not accept another incoming connection until an existing connection is released. Explain how semaphores can be used by a server to limit the number of concurrent connections.

There’s only as many items for the semaphore as there are desired open connections. A new connection cannot be opened if the semaphore cannot be used. Ideally this would result in a buffer for some ports to be used to cycle connections.